

**IN THE UNITED STATES DISTRICT COURT  
FOR THE EASTERN DISTRICT OF TEXAS  
MARSHALL DIVISION**

**RYAN HARDIN AND  
ANDREW HILL,**

**Plaintiffs,**

**vs.**

**SAMSUNG ELECTRONICS CO., LTD., AND  
SAMSUNG ELECTRONICS AMERICA, INC.,**

**Defendants.**

**Civil Action No. 2:21-cv-290**

**JURY TRIAL**

**COMPLAINT FOR PATENT INFRINGEMENT**

Plaintiffs Ryan Hardin and Andrew Hill file this Complaint for Patent Infringement against Samsung Electronics Co., Ltd. and Samsung Electronics America, Inc. (collectively, “Samsung”), and allege as follows:

**THE PARTIES**

1. Plaintiff Ryan Hardin is a citizen of Texas with a principal residence in Houston, Texas. Mr. Hardin is a technology professional with over 20 years of experience.
2. Plaintiff Andrew Hill is a citizen of Texas with a principal residence in Houston, Texas. Mr. Hill is a technology professional with over 20 years of experience.
3. Defendant Samsung Electronics Co., Ltd. (“SEC”) is a Korean company with its principal place of business in Suwon, South Korea. SEC has an “Information Technology & Mobile Communications” division that is responsible for the design, manufacture, and sale of

cellular network infrastructure equipment and components thereof around the world and in the United States.

4. Defendant Samsung Electronics America, Inc. (“SEA”) is a New York corporation with its principal place of business in Ridgefield Park, New Jersey and an established place of business in Plano, Texas. SEA is a wholly-owned subsidiary of SEC. SEA imports into the United States and sells in the United States, including in this District, cellular network infrastructure equipment and components thereof.

### **JURISDICTION AND VENUE**

5. This is an action arising under the patent laws of the United States, 35 U.S.C. § 271. This Court has subject matter jurisdiction under 28 U.S.C. §§ 1331 and 1338(a).

6. Venue is proper in this judicial district under 28 U.S.C. §§ 1391 and 1400(b) because Defendant has committed acts of infringement and has a regular and established place of business in this District. SEA, SEC’s wholly-owned subsidiary, maintains an office in Plano, Texas. (See “Where to Find Us,” Samsung, <https://www.sra.samsung.com/locations/>). In 2018, SEA relocated its North Texas-based teams in Richardson to Plano to Legacy Central in Plano. SEA described this new Plano office as its “flagship North Texas campus” and “home to Samsung Electronics America’s second biggest employee population in the U.S. across multiple divisions – Customer Care, Mobile, Mobile R&D and Engineering.” (See “Samsung Electronics America to Open Flagship North Texas Campus,” Samsung Newsroom U.S., <https://news.samsung.com/us/samsung-electronics-america-open-flagship-north-texas-campus/> (last visited July 27, 2021)). In a press release, SEA said its “move to Plano further demonstrates the company’s dedication to Texas and being an invested corporate citizen.” (*Id.*)

7. This Court has personal jurisdiction over Samsung. Samsung has continuous and systematic business contacts with the State of Texas. Samsung, directly and/or through subsidiaries or intermediaries (including distributors, retailers, and others), conducts its business extensively throughout Texas, by shipping, distributing, offering for sale, selling, and advertising (including the provision of an interactive web page) its products and/or services in the State of Texas and the Eastern District of Texas. SEA maintains an office in Plano, Texas and is responsible for importing and selling smartphones, tablets, other mobile devices, and cellular network infrastructure equipment that operate on cellular networks in the United States. SEC and SEA regularly do business or solicit business, engage in other persistent courses of conduct, and/or derive substantial revenue from products and/or services provided to individuals in the State of Texas.

8. SEC and SEA, directly and/or through subsidiaries or intermediaries (including distributors, retailers, and others), have purposefully and voluntarily placed one or more products and/or services in the stream of commerce that practice the Asserted Patents with the intention and expectation that they will be purchased and used by consumers in the Eastern District of Texas. These products and/or services have been and continue to be purchased and used by operators/carriers in the Eastern District of Texas.

9. Upon information and belief, the Samsung products accused of infringement in this case are manufactured by or on behalf of SEC and SEA.

### **BACKGROUND OF THE INVENTORS**

10. Inventors Ryan Hardin and Andrew Hill both grew up in Henderson, Texas. After college, Hardin and Hill started an IT services company called Pronet Solutions Corp. in 2002 in East Texas. Upon establishing the business after a few years, Hardin and Hill began to

investigate and discuss emerging and future business opportunities that would leverage their technological backgrounds and expertise.

11. In the 2006-2007 timeframe, Hardin and Hill were interested in the digital billboard business. At that time, digital billboards were relatively new, and Hardin and Hill saw an opportunity to grow a business for displaying content (e.g., advertisements) on digital billboards, which they could efficiently manage and control remotely. However, Hardin and Hill ultimately saw significant barriers to entry for getting into the digital billboard business, including costs, regulatory considerations, and governmental issues.

12. Then, in late 2007, Hardin was at a meeting in Henderson, and when getting in his car to go to his next appointment, he looked at his phone and thought—why should he and Hill risk the expense to erect digital billboards if much of the public might soon be carrying digital billboards in their pockets? Of course, at that time, smartphones were in a period of rapid development,<sup>1</sup> and while a few phones had GPS, access to the internet, and apps, they were certainly not as commonplace as today. Hardin and Hill were betting on the future. Going forward, Hardin and Hill continued to fine-tune the implementation of their invention and applied for their first patent on May 1, 2009.

13. Separate from the patented invention, not only did Hardin and Hill realize a technical solution was needed—but a platform was also needed for the technology to be marketed to consumers. In this same timeframe, Hardin and Hill coincidentally received a phone call from

---

<sup>1</sup> For example, upon information and belief, the first version of the Samsung Galaxy smartphone was released in June 2009, and Google launched the Android Market in October 2008 as a way for users to download apps for Android-based phones (e.g., Samsung Galaxy).

the Henderson Area Chamber of Commerce, which requested a website be designed—but as cheaply as possible. Hardin and Hill ultimately saw this as an opportunity for the platform they were looking for. As a result, Hardin and Hill decided they would develop and host the website for free for the Henderson Area Chamber of Commerce, but then ultimately, build free websites for all Chambers of Commerce using the same platform. Then, through the access the chambers had with businesses, the websites could get advertising revenue—which is how Hardin and Hill could ultimately be compensated. Moreover, once setting up this network of chambers of commerce was complete, eventually the patented technology could also be marketed and used to create location-aware content delivery on mobile devices as they envisioned an increased demand from businesses for location-aware apps and from the public for mobile devices. Hardin and Hill called the platform “Chamber Planet.”

14. Hardin and Hill spent years developing Chamber Planet and started to get numerous chambers of commerce on board. At one point, Hardin and Hill were working with the Henderson Area Chamber, Jacksonville Chamber, Bullard Area Chamber, Pittsburgh Camp County Chamber, and the Greater Marshall Chamber of Commerce. In fact, Hardin and Hill negotiated the purchase of the MarshallTexas.com domain from a third party and provided it to the Greater Marshall Chamber of Commerce. As more chambers got on board, more challenges arose, including that the chambers also desired an integrated billing system with their websites—which was complex and took significant time to develop.

15. On or around 2012, however, while Hardin and Hill were still developing and growing the Chamber Planet platform, their separate business Pronet also began to experience some challenges, largely due to the business disruptions created by the financial crisis. Hardin and

Hill had to focus more on Pronet, which was the business that provided revenue to them—and this slowed the growth of Chamber Planet. In addition, numerous members of Hardin’s family began to have serious health issues. Hardin’s wife had medical issues in pregnancies which required multiple surgeries and then she later developed cancer. Around the same time, Hardin’s mother fell and broke her back, and Hardin’s father required a lung transplant. Hardin needed to prioritize his family over the continued development of Chamber Planet for a period of time.

16. During this period of time, however, Hardin and Hill noticed that mobile devices were starting to utilize Hardin and Hill’s invention. Moreover, these companies developing the mobile devices and related software and services had significantly more financial resources that enabled them to commercialize the invention much more quickly than Hardin or Hill could. As a result, Hardin and Hill then decided to focus more on protecting the intellectual property they had developed, while continuing to look out for opportunities to bounce back and continue to build the platform they had invented.

### **THE ASSERTED PATENTS**

17. Plaintiffs are the owners in right, title, and interest in and to multiple United States patents and patent applications, including U.S. Patent No. 9,779,418 (the “’418 Patent”), U.S. Patent No. 10,049,387 (the “’387 Patent”), and U.S. Patent No. 10,984,447 (the “’447 Patent”) (collectively, the “Asserted Patents”). The Asserted Patents are valid and enforceable, and the inventions claimed in the Asserted Patents were novel, non-obvious, unconventional, and non-routine at least as of May 1, 2009.

18. The ’418 Patent, entitled “Exclusive Delivery of Content within Geographic Areas,” was duly and legally issued to inventors Ryan Hardin and Andrew Hill on October 3, 2017.

Plaintiffs own the entire right, title, and interest in the '418 Patent and are entitled to sue for past and future infringement. A true and correct copy of the '418 Patent is attached as Exhibit A.

19. The '387 Patent, entitled "Exclusive Delivery of Content within Geographic Areas," was duly and legally issued to inventors Ryan Hardin and Andrew Hill on August 14, 2018. Plaintiffs own the entire right, title, and interest in the '387 Patent and are entitled to sue for past and future infringement. A true and correct copy of the '387 Patent is attached as Exhibit B.

20. The '447 Patent, entitled "Exclusive Delivery of Content within Geographic Areas," was duly and legally issued to inventors Ryan Hardin and Andrew Hill on April 20, 2021. Plaintiffs own the entire right, title, and interest in the '447 Patent and are entitled to sue for past and future infringement. A true and correct copy of the '447 Patent is attached as Exhibit C.

### **OVERVIEW OF THE INVENTIONS**

21. Mobile applications ("mobile apps" or "apps") are computer programs designed to run on mobile devices. The advent and proliferation of apps for mobile devices significantly increased the opportunities for providing content to device users. In particular, the ability of mobile phones to monitor and make decisions based on the dynamic location of the user created additional opportunities for providing location-based content, including advertisements and other data or information.

22. Apps had no precise analog in physical media such as newspapers or magazines. Nor did physical media have the ability to determine a person's location and proximity to other points of interest at any given time, such as when a user is moving from one location to another, or the length of time that a user spends in a certain location. Indeed, the location of a mobile

device may change rapidly depending on, for example, whether the user is stationary or in a moving vehicle.

23. With these new opportunities came new technological challenges, including how to enable location-dependent features and functionality for apps on a mobile device while constrained by the limited computing power and storage of a mobile device. For instance, location information can be obtained by apps on a mobile device in a number of suitable ways, including using a Global Positioning System (GPS) component on the mobile device. One way would be for each app to communicate with the GPS component to determine the location of the mobile device and deliver location-dependent content. Each app requiring location information would need to be configured to independently obtain location information. As such, if, for example, 10 mobile apps polled a GPS component for a location, the GPS component would need to obtain location information (e.g., through satellites) 10 different times.

24. However, given that mobile devices may include dozens of apps installed and simultaneously running on a mobile device, having apps independently requesting location information would be processor intensive and not scalable, especially for the mobile device to efficiently allow multitasking when several apps are simultaneously running in the foreground/background. This implementation could tie up the limited resources available on the mobile phone and also deplete the mobile phone's battery at a faster rate.

25. Plaintiffs invented a technological solution that solves these technical problems by implementing a centralized service on a mobile device for tracking and monitoring the location of the mobile device on behalf of apps on the mobile device. Apps can register with the centralized service to receive its benefits. The claimed inventions provide a mechanism whereby an app can



register with the centralized service and request to be notified about events relating to a particular geographic area of interest, which is defined by a virtual perimeter. One kind of geographic area of interest is now commonly referred to as a “geofence” (the term is used herein as an exemplary geographic area of interest but not to suggest exact equivalence of the terms). As part of the request to be notified about events relating to a particular geofence, an app provides a unique identifier corresponding to the app and a geofence. The centralized service receives and stores identifiers to keep track of app/geofence pairings for registered apps. The centralized service monitors the location of the mobile device and alerts the appropriate app(s), based on the identifier, when there is an event associated with a particular geofence (e.g., when the mobile device enters or exits a geofence registered to one or more apps on the mobile device, when the mobile device remains in a geofence for a predetermined amount of time). The alert from the centralized service includes the identifier provided by the app. Apps can advantageously limit the frequency of alerts received from the centralized service by increasing the amount of time the mobile device must remain within a particular area before an alert is triggered. In response to the alert, the app can, for example, display location-based information for the user. As such, for example, individual apps are able to customize information that is shown based on the mobile device’s location, including based on the mobile device moving to a different location and/or the duration it remains within a particular area. Notably, defining alerts and triggering content delivery based on geofence events was not feasible for physical media such as newspapers and the like, which cannot respond to dynamic movement.

26. By offloading the polling and monitoring of the mobile device’s location from the individual apps to the centralized service, the claimed inventions improve the overall functioning

of the mobile devices as well as the individual apps. As a result, the claimed inventions improve the overall functioning of the mobile device itself by increasing processing efficiency and reducing battery drain, as well as the functioning of the individual apps, especially when several apps are running simultaneously in the foreground or background. Revisiting the example described above, the centralized service could meet the needs of 10 apps by obtaining location information once for all of the apps. The centralized service, which is separate from the apps on the mobile device, could be a part of the operating system (OS) of the mobile device or it may be a separate component of the mobile device. By using a centralized service, the individual apps can take advantage of any additional information known to the mobile device (e.g., sensor data such as an accelerometer), thereby improving the functionality of the device and apps.

**COUNT I: CLAIM FOR PATENT INFRINGEMENT OF THE '418 PATENT**

27. Plaintiffs repeat and reallege the allegations in paragraphs 1-26 as if fully set forth herein.

28. Samsung has infringed, contributed to the infringement of, and/or induced infringement of the '418 Patent by making, using, selling, offering for sale, or importing into the United States, or by intending that others make, use, import into, offer for sale, or sell in the United States, products and/or methods covered by one or more claims of the '418 Patent including, but not limited to, mobile devices and components thereof.

29. Upon information and belief, Samsung's mobile devices, including smartphones, tablets, and watches, that infringe one or more claims of the '418 Patent include at least the following products, as well as products with reasonably similar functionality:

- Galaxy S series, Galaxy Note series, Galaxy Z Flip series, Galaxy Fold series, and Galaxy A series smartphones;
- Galaxy Tab S series, Galaxy Tab A series, and Galaxy Book S series tablets; and
- Galaxy Watch 3 series, Galaxy Watch 3 Titanium series, Galaxy Watch Active2 series, Galaxy Watch Active series, Galaxy Watch series, and Galaxy Fit2 series watches.

Upon information and belief, these products are offered in several varieties, including different options for size, display, processor, storage, and battery. The products listed are exemplary, and Plaintiffs will be able to provide a more comprehensive list after discovery. Upon information and belief, Samsung also infringes one or more claims of the '418 Patent through its use of geofencing or location information in its advertising or content delivery efforts or its own applications it develops.

30. For example, upon information and belief, Samsung's mobile devices infringe at least claim 11 of the '418 Patent. Samsung makes, uses, sells, offers for sale, imports, exports, supplies, or distributes within the United States its mobile devices and thus directly infringes the '418 Patent.

31. Samsung indirectly infringes the '418 Patent as provided by 35 U.S.C. § 271(b) by inducing infringement by others, such as manufacturers, resellers, and end-user customers in this District and throughout the United States. For example, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices, who, upon information and belief, perform each step of the claimed invention as directed

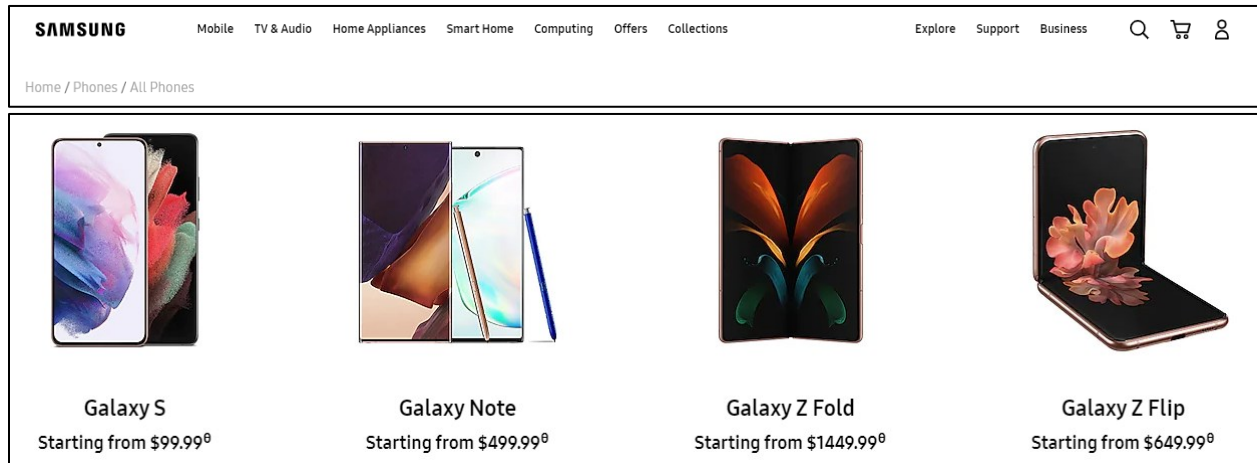
by Samsung. Samsung received actual notice of the '418 Patent at least as early as the filing of this Complaint.

32. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices, causing Samsung's mobile devices to be manufactured, and providing directions, instructions, schematics, diagrams, or designs to its manufacturers, resellers, or operators/carriers to make or use Samsung's mobile devices in a manner that directly infringes the '418 Patent induce infringement by others, such as manufacturers, resellers, or operators/carriers in this District and throughout the United States. Upon information and belief, through its manufacture and sales of Samsung's mobile devices, Samsung performed the acts that constitute induced infringement with knowledge or willful blindness that the induced acts would constitute infringement.

33. Samsung also indirectly infringes the '418 Patent by contributing to infringement by others, such as manufacturers, resellers, manufacturers, and operators/carriers, in accordance with 35 U.S.C. § 271(c) in this District and throughout the United States. Upon information and belief, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices.

34. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices and causing Samsung's mobile devices to be manufactured and sold contribute to Samsung's manufacturers, resellers, and operators/carriers making or using Samsung's mobile devices in a normal and customary way that infringes the '418 Patent. Upon information and belief, Samsung's mobile devices constitute the material part of Plaintiffs' patented invention, have no substantial non-infringing uses, and are known by Samsung to be especially made or especially adapted for use to infringe the '418 Patent.

35. Samsung provides a mobile device comprising: memory, at least one processor operably coupled to the memory, a location-determination component tangibly embodied in the mobile device for determining the mobile device's physical geographic location, and at least one module stored in the memory and configured for execution by the at least one processor, including as demonstrated in the exemplary text and images below:




<https://www.samsung.com/us/>

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	COMPARE
Memory	<b>Galaxy S10e</b>	<ul style="list-style-type: none"> <li>• 6GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 256GB internal storage</li> </ul>	
	<b>Galaxy S10</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	
	<b>Galaxy S10+</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage (Only ceramic models)</li> <li>• 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)</li> </ul>	
	<b>Galaxy S10 5G</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 256GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	COMPARE
AP		8nm 64-bit Octa-Core Processor ※ 2.73GHz (Maximum Clock Speed) + 2.31GHz + 1.95GHz	
		7nm 64-bit Octa-Core Processor ※ 2.84GHz (Maximum Clock Speed) + 2.41GHz + 1.78GHz	
		*May differ by country and carrier.	

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
OS	Android 9 (Pie)		

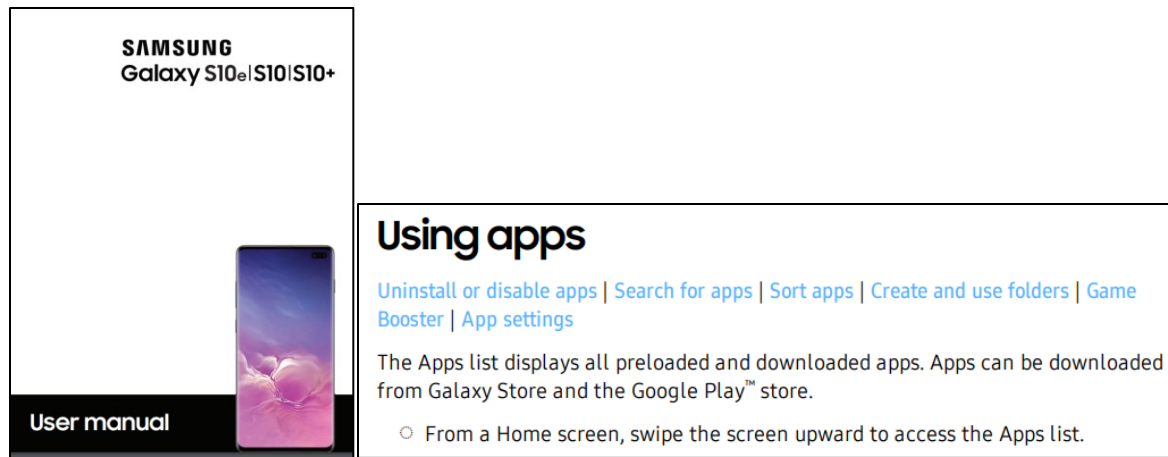
SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Network & Connectivity	<p>Enhanced 4x4 MIMO, Up to 7CA, LAA, LTE Cat.20 Up to 2.0Gbps Download / Up to 150Mbps Upload</p> <p>5G Non-Standalone (NSA), Sub6 / mmWave (Galaxy S10 5G only)</p> <p>Wi-Fi 802.11 a/b/g/n/ac/ax (2.4/5GHz), VHT80 MU-MIMO, 1024QAM Up to 1.2Gbps Download / Up to 1.2Gbps Upload</p> <p>Bluetooth® v 5.0 (LE up to 2Mbps), ANT+, USB type-C, NFC, Location (GPS, Galileo, Glonass, BeiDou)</p> <p><small>*Actual speed may vary depending on country, carrier, and user environment. *Depending on country or carrier, 5G connectivity may use Sub6 only, mmWave only, or Sub6 and mmWave together. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries.</small></p>		

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Bixby 	<p>Bixby Routines</p> <p>Bixby</p> <p>Bixby Vision</p> <p>Apps Mode</p> <ul style="list-style-type: none"> <li>• Home décor</li> <li>• Makeup &amp; Hair Dyeing</li> <li>• Styling</li> </ul> <p>Lens Mode</p> <ul style="list-style-type: none"> <li>• Image</li> <li>• Shopping</li> <li>• Translation</li> <li>• Place</li> </ul> <ul style="list-style-type: none"> <li>• Picture play</li> <li>• 3rd party Downloads</li> <li>• Food</li> <li>• Wine</li> <li>• Book</li> <li>• QR Code</li> </ul>		

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

36. Samsung provides a mobile device, wherein the at least one module comprises instructions for: implementing, on the mobile device, at least one computer readable program

instruction, made available for use by one or more distinct application programs operating on the mobile device, for receiving, from a particular one application program of the one or more distinct application programs during the particular one application program's execution on the mobile device, a request to reserve, for the particular one application program, at least one designated geographic area of interest for having a particular associated identifier exclusively provided for use by the particular one application program after it is determined, by at least use of the at least one processor and of location information representing a physical geographic location of the mobile device as determined by the location-determination component, that the mobile device has at least entered the at least one designated geographic area of interest, wherein the at least one computer readable program instruction comprises at least one input parameter for receiving, from the particular one application program, data representing a) the particular associated identifier as provided content associated with the at least one designated geographic area of interest and exclusively related to the particular one application program, and b) a perimeter definition defining, with at least reference to longitude and latitude, the at least one designated geographic area of interest, and wherein the one or more distinct application programs each comprises a system of executable coded instructions as programmed by at least one developer for execution on the mobile device, including as demonstrated in the exemplary text and images below:



[https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT\\_SM-G970U\\_G973U\\_G975U\\_EN\\_UM\\_R\\_11.0\\_022221\\_FINAL.pdf](https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT_SM-G970U_G973U_G975U_EN_UM_R_11.0_022221_FINAL.pdf)



## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Set up for geofence monitoring

The first step in requesting geofence monitoring is to request the necessary permissions. To use geofencing, your app must request the following:

- `ACCESS_FINE_LOCATION`
- `ACCESS_BACKGROUND_LOCATION` if your app targets Android 10 (API level 29) or higher

To learn more, see the guide on how to [request location permissions](#).

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
    android:allowBackup="true">
    ...
    <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
    android:allowBackup="true">
    ...
    <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

To access the location APIs, you need to create an instance of the Geofencing client. To learn how to connect your client:

Kotlin      Java

```
lateinit var geofencingClient: GeofencingClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this)
}
```

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
            Java

geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin   Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```

geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>

37. Samsung provides a mobile device, wherein the at least one module comprises instructions for: receiving, using the at least one computer readable program instruction, from a first particular application program during its execution on the mobile device, a first request for a reservation for a first particular designated geographic area of interest for having a first particular associated identifier exclusively provided for use by the first particular application program after it is determined that the mobile device has at least entered the first particular designated geographic area of interest, wherein the first request comprises data representing a) the first particular associated identifier, represented by a data string, as provided content by the first particular application program to be associated with the first particular designated geographic area of interest and exclusively related to the first particular application program, and b) a first perimeter definition

provided by the first particular application program, wherein the first perimeter definition comprises at least data representing a first latitude and longitude coordinate and a first radius value for defining the first particular designated geographic area of interest, including as demonstrated in the exemplary text and images below:

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
            Java

geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.



## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```

geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>

38. Samsung provides a mobile device, wherein the at least one module comprises instructions for: determining availability for the first particular designated geographic area of interest to be reserved for the first particular application program, including as demonstrated in the exemplary text and images below:

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

## GeofenceStatusCodes

public final class **GeofenceStatusCodes** extends `CommonStatusCodes`

Geofence specific status codes, for use in `Status.getStatusCode()`

### Constant Summary

int	<code>GEOFENCE_INSUFFICIENT_LOCATION_PERMISSION</code>	The client doesn't have sufficient location permission to perform geofencing operations.
int	<code>GEOFENCE_NOT_AVAILABLE</code>	Geofence service is not available now.
int	<code>GEOFENCE_REQUEST_TOO_FREQUENT</code>	Your app has been adding Geofences too frequently.
int	<code>GEOFENCE_TOO_MANY_GEOFENCES</code>	Your app has registered more than 100 geofences.
int	<code>GEOFENCE_TOO_MANY_PENDING_INTENTS</code>	You have provided more than 5 different PendingIntents to the <code>GeofencingApi.addGeofences(GoogleApiClient, GeofencingRequest, PendingIntent)</code> call.


<https://developers.google.com/android/reference/com/google/android/gms/location/GeofenceStatusCodes>

39. Samsung provides a mobile device, wherein the at least one module comprises instructions for: using the first perimeter definition in conjunction with the first particular associated identifier to establish the first particular designated geographic area of interest in the first request as an area reserved for first content related to the first particular application program and not an area reserved for first content related to other application programs, wherein the first content related to the first particular application program comprises the first particular associated identifier, including as demonstrated in the exemplary text and images below:

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a [PendingIntent](#) as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use [Geofence.Builder](#) to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin  Java
geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```

geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>

40. Samsung provides a mobile device, wherein the at least one module comprises instructions for: storing, in the memory, in response to receiving the first request, after availability for the first particular designated geographic area of interest to be reserved for the first particular application program has been positively determined, an at least first record, for the first particular application program, associated with the first particular associated identifier and the first particular designated geographic area of interest, thereby designating the first particular application program as a first one of the one or more registered application programs on the mobile device, wherein the one or more registered application programs on the mobile device each consists of an application program designated for having a particular identifier, being associated with a particular reserved geographic area of interest, to be exclusively provided for use by said application program after it

is determined, by at least use of the at least one processor and of the location information, that the mobile device has at least entered said particular reserved geographic area of interest, including as demonstrated in the exemplary text and images below:

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Memory	<b>Galaxy S10e</b>		<b>Galaxy S10</b>
	<ul style="list-style-type: none"> <li>• 6GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 256GB internal storage</li> </ul>		<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>
	<b>Galaxy S10+</b>		<b>Galaxy S10 5G</b>
	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage (Only ceramic models)</li> <li>• 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)</li> </ul>		<ul style="list-style-type: none"> <li>• 8GB RAM with 256GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

### Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

```

Kotlin  Java
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>



## GeofencingClient

public class **GeofencingClient** extends [GoogleApi<Api.ApiOptions.NoOptions>](#)

The main entry point for interacting with the geofencing APIs.

Get an instance of this client via [LocationServices.getGeofencingClient\(Activity\)](#).

All methods are thread safe.

### Public Method Summary

Task<Void>	<a href="#">addGeofences(GeofencingRequest geofencingRequest, PendingIntent pendingIntent)</a> Sets alerts to be notified when the device enters or exits one of the specified geofences.
Task<Void>	<a href="#">removeGeofences(List&lt;String&gt; geofenceRequestIds)</a> Removes geofences by their request IDs.
Task<Void>	<a href="#">removeGeofences(PendingIntent pendingIntent)</a> Removes all geofences associated with the given pendingIntent.

**public Task<Void> addGeofences** ([GeofencingRequest geofencingRequest](#), [PendingIntent pendingIntent](#))

Sets alerts to be notified when the device enters or exits one of the specified geofences. If an existing geofence with the same request ID is already registered, the old geofence is replaced by the new one, and the new [PendingIntent](#) is used to generate intents for alerts.

[Task](#) is completed when geofences are successfully added or failed to be added. Upon failure, an [ApiException](#) will be set on the [Task](#). Refer to [GeofenceStatusCodes](#) for possible errors when adding geofences.

When a geofence transition (for example, entering or exiting) matches one of the transition filter (see [Geofence.Builder.setTransitionTypes\(int\)](#)) in the given geofence list, an intent is generated using the given pending intent. You can call [GeofencingEvent.fromIntent\(android.content.Intent\)](#) to get the transition type, geofences that triggered this intent and the location that triggered the geofence transition.

In case network location provider is disabled by the user, the geofence service will stop updating, all registered geofences will be removed and an intent is generated by the provided pending intent. In this case, the [GeofencingEvent](#) created from this intent represents an error event, where [GeofencingEvent.hasError\(\)](#) returns `true` and [GeofencingEvent.getErrorCode\(\)](#) returns [GeofenceStatusCodes.GEOFENCE\\_NOT\\_AVAILABLE](#).

This method requires [Manifest.permission.ACCESS\\_FINE\\_LOCATION](#).

#### Parameters

<b>geofencingRequest</b>	geofencing request that include a list of geofences to be added and related triggering behavior. The request must be created using <a href="#">GeofencingRequest.Builder</a> .
<b>pendingIntent</b>	a pending intent that will be used to generate an intent when matched geofence transition is observed

#### Throws

<a href="#">SecurityException</a>	if the app does not have <a href="#">Manifest.permission.ACCESS_FINE_LOCATION</a> permission
<a href="#">NullPointerException</a>	if geofencingRequest or pendingIntent is null

<https://developers.google.com/android/reference/com/google/android/gms/location/GeofencingClient.html>

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

Kotlin    Java

```
class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}
```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

<https://developer.android.com/training/location/geofencing>

41. Samsung provides a mobile device, wherein the at least one module comprises instructions for: receiving, using the at least one computer readable program instruction, from a second particular application program during its execution on the mobile device, a second request for a reservation for a second particular designated geographic area of interest for having a second particular associated identifier exclusively provided for use by the second particular application program after it is determined that the mobile device has at least entered the second particular designated geographic area of interest, wherein the second request comprises data representing a) the second particular associated identifier, represented by a data string, as provided content by the second particular application program to be associated with the second particular designated geographic area of interest and exclusively related to the second particular application program, and b) a second perimeter definition provided by the second particular application program, wherein the second perimeter definition comprises at least data representing a second latitude and longitude coordinate and a second radius value for defining the second particular designated geographic area of interest, including as demonstrated in the exemplary text and images below:

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
Geofence.Builder()
// Set the request ID of the geofence. This is a string to identify this
// geofence.
.setRequestId(entry.getKey())

.setCircularRegion(
    entry.getValue().latitude,
    entry.getValue().longitude,
    Constants.GEOFENCE_RADIUS_IN_METERS
)
.setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
.setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
    Geofence.GEOFENCE_TRANSITION_EXIT)
.build();
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin   Java

```
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(geofenceList);
    return builder.build();
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

42. Samsung provides a mobile device, wherein the at least one module comprises instructions for: determining availability for the second particular designated geographic area of interest to be reserved for the second particular application program, including as demonstrated in the exemplary text and images below:

### Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin

Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

## GeofenceStatusCodes

public final class **GeofenceStatusCodes** extends [CommonStatusCodes](#)

Geofence specific status codes, for use in [Status.getStatusCode\(\)](#)

### Constant Summary

int	<a href="#">GEOFENCE_INSUFFICIENT_LOCATION_PERMISSION</a>	The client doesn't have sufficient location permission to perform geofencing operations.
int	<a href="#">GEOFENCE_NOT_AVAILABLE</a>	Geofence service is not available now.
int	<a href="#">GEOFENCE_REQUEST_TOO_FREQUENT</a>	Your app has been adding Geofences too frequently.
int	<a href="#">GEOFENCE_TOO_MANY_GEOFENCES</a>	Your app has registered more than 100 geofences.
int	<a href="#">GEOFENCE_TOO_MANY_PENDING_INTENTS</a>	You have provided more than 5 different PendingIntents to the <a href="#">GeofencingApi.addGeofences(GoogleApiClient, GeofencingRequest, PendingIntent)</a> call.

<https://developers.google.com/android/reference/com/google/android/gms/location/GeofenceStatusCodes>

43. Samsung provides a mobile device, wherein the at least one module comprises instructions for: using the second perimeter definition in conjunction with the second particular associated identifier to establish the second particular designated geographic area of interest in the second request as an area reserved for second content related to the second particular application program and not an area reserved for second content related to other application programs, wherein the second content related to the second particular application program comprises the second particular associated identifier, including as demonstrated in the exemplary text and images below:



## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin  Java
geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin   Java

```
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(geofenceList);
    return builder.build();
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

44. Samsung provides a mobile device, wherein the at least one module comprises instructions for: storing, in the memory, in response to receiving the second request, after availability for the second particular designated geographic area of interest to be reserved for the second particular application program has been positively determined, an at least second record, for the second particular application program, associated with the second particular associated identifier and the second particular designated geographic area of interest, thereby designating the second particular application program as a second one of the one or more registered application programs on the mobile device, including as demonstrated in the exemplary text and images below:

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Memory	<b>Galaxy S10e</b>	<b>Galaxy S10</b>	
	<ul style="list-style-type: none"> <li>• 6GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 256GB internal storage</li> </ul>	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	
	<b>Galaxy S10+</b>	<b>Galaxy S10 5G</b>	
	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage (Only ceramic models)</li> <li>• 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)</li> </ul>	<ul style="list-style-type: none"> <li>• 8GB RAM with 256GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

## GeofencingClient

public class **GeofencingClient** extends `GoogleApi<Api.ApiOptions.NoOptions>`

The main entry point for interacting with the geofencing APIs.

Get an instance of this client via `LocationServices.getGeofencingClient(Activity)`.

All methods are thread safe.

### Public Method Summary

Task<Void>	<code>addGeofences(GeofencingRequest geofencingRequest, PendingIntent pendingIntent)</code> Sets alerts to be notified when the device enters or exits one of the specified geofences.
Task<Void>	<code>removeGeofences(List&lt;String&gt; geofenceRequestIds)</code> Removes geofences by their request IDs.
Task<Void>	<code>removeGeofences(PendingIntent pendingIntent)</code> Removes all geofences associated with the given <code>pendingIntent</code> .

```
public Task<Void> addGeofences (GeofencingRequest geofencingRequest, PendingIntent pendingIntent)
```

Sets alerts to be notified when the device enters or exits one of the specified geofences. If an existing geofence with the same request ID is already registered, the old geofence is replaced by the new one, and the new `PendingIntent` is used to generate intents for alerts.

`Task` is completed when geofences are successfully added or failed to be added. Upon failure, an `ApiException` will be set on the `Task`. Refer to `GeofenceStatusCodes` for possible errors when adding geofences.

When a geofence transition (for example, entering or exiting) matches one of the transition filter (see `Geofence.Builder.setTransitionTypes(int)`) in the given geofence list, an intent is generated using the given pending intent. You can call `GeofencingEvent.fromIntent(android.content.Intent)` to get the transition type, geofences that triggered this intent and the location that triggered the geofence transition.

In case network location provider is disabled by the user, the geofence service will stop updating, all registered geofences will be removed and an intent is generated by the provided pending intent. In this case, the `GeofencingEvent` created from this intent represents an error event, where `GeofencingEvent.hasError()` returns `true` and `GeofencingEvent.getErrorCode()` returns `GeofenceStatusCodes.GEOFENCE_NOT_AVAILABLE`.

This method requires `Manifest.permission.ACCESS_FINE_LOCATION`.

#### Parameters

<code>geofencingRequest</code>	geofencing request that include a list of geofences to be added and related triggering behavior. The request must be created using <code>GeofencingRequest.Builder</code> .
<code>pendingIntent</code>	a pending intent that will be used to generate an intent when matched geofence transition is observed

#### Throws

<code>SecurityException</code>	if the app does not have <code>Manifest.permission.ACCESS_FINE_LOCATION</code> permission
<code>NullPointerException</code>	if <code>geofencingRequest</code> or <code>pendingIntent</code> is null

<https://developers.google.com/android/reference/com/google/android/gms/location/GeofencingClient.html>

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin      Java
class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

<https://developer.android.com/training/location/geofencing>

45. Samsung provides a mobile device, wherein the at least one module comprises instructions for: selecting the first particular associated identifier associated with the at least first record stored in the memory to be provided for use by the first one of the one or more registered application programs on the mobile device after it is determined, by at least use of the at least one processor, of the location information, and of the first particular designated geographic area of interest associated with the at least first record stored in the memory for the first one of the one or more registered application programs on the mobile device, that the mobile device has at least entered the first particular designated geographic area of interest associated with the at least first record stored in the memory and has remained therein for at least a first designated length of time, including as demonstrated in the exemplary text and images below:



## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin    Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin  Java

class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

**Use the dwell transition type to reduce alert spam**

If you receive a large number of alerts when driving briefly past a geofence, the best way to reduce the alerts is to use a transition type of `GEOFENCE_TRANSITION_DWELL` instead of `GEOFENCE_TRANSITION_ENTER`. This way, the dwelling alert is sent only when the user stops inside a geofence for a given period of time. You can choose the duration by setting a [loitering delay](#).

<https://developer.android.com/training/location/geofencing>

46. Samsung provides a mobile device, wherein the at least one module comprises instructions for: selecting the second particular associated identifier associated with the at least second record stored in the memory to be provided for use by the second one of the one or more registered application programs on the mobile device after it is determined, by at least use of the at least one processor, of the location information, and of the second particular designated geographic area of interest associated with the at least second record stored in the memory for the second one of the one or more registered application programs on the mobile device, that the mobile device has at least entered the second particular designated geographic area of interest associated with the at least second record stored in the memory and has remained therein for at least a second designated length of time, including as demonstrated in the exemplary text and images below:

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin    Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin  Java

class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

### Use the dwell transition type to reduce alert spam

If you receive a large number of alerts when driving briefly past a geofence, the best way to reduce the alerts is to use a transition type of `GEOFENCE_TRANSITION_DWELL` instead of `GEOFENCE_TRANSITION_ENTER`. This way, the dwelling alert is sent only when the user stops inside a geofence for a given period of time. You can choose the duration by setting a `loitering delay`.

<https://developer.android.com/training/location/geofencing>

47. Samsung provides a mobile device, wherein the at least one module comprises instructions for: providing, to the first one of the one or more registered application programs on the mobile device, at least the first particular associated identifier from its selecting in response to receiving, from the first one of the one or more registered application programs on the mobile device, a request to receive content, including as demonstrated in the exemplary text and images below:

### Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

```
Kotlin  Java
public class MainActivity extends AppCompatActivity {
    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```



## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin      Java

class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

<https://developer.android.com/training/location/geofencing>

48. Samsung provides a mobile device, wherein the at least one module comprises instructions for: providing, to the second one of the one or more registered application programs on the mobile device, at least the second particular associated identifier from its selecting in response to receiving, from the second one of the one or more registered application programs on the mobile device, a request to receive content, including as demonstrated in the exemplary text and images below:

### Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

Kotlin

Java

```
class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}
```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

<https://developer.android.com/training/location/geofencing>

49. By engaging in the conduct described herein, Samsung has injured Plaintiffs and is thus liable for infringement of the '418 Patent, pursuant to 35 U.S.C. § 271. Samsung has committed these acts of infringement without license or authorization.

50. As a result of Samsung's infringement of the '418 Patent, Plaintiffs have suffered monetary damages and are entitled to a monetary judgment in an amount adequate to compensate for Samsung's past infringement, together with interests and costs. In addition, Samsung's infringement is causing irreparable harm and monetary damage to Plaintiffs and will continue to do so unless and until Samsung is enjoined by the Court.

**COUNT II: CLAIM FOR PATENT INFRINGEMENT OF THE '387 PATENT**

51. Plaintiffs repeat and reallege the allegations in paragraphs 1-50 as if fully set forth herein.

52. Samsung has infringed, contributed to the infringement of, and/or induced infringement of the '387 Patent by making, using, selling, offering for sale, or importing into the United States, or by intending that others make, use, import into, offer for sale, or sell in the United States, products and/or methods covered by one or more claims of the '387 Patent including, but not limited to, mobile devices and components thereof.

53. Upon information and belief, Samsung's mobile devices, including smartphones, tablets, and watches, that infringe one or more claims of the '387 Patent include at least the following products, as well as products with reasonably similar functionality:

- Galaxy S series, Galaxy Note series, Galaxy Z Flip series, Galaxy Fold series, and Galaxy A series smartphones;

- Galaxy Tab S series, Galaxy Tab A series, and Galaxy Book S series tablets; and
- Galaxy Watch 3 series, Galaxy Watch 3 Titanium series, Galaxy Watch Active2 series, Galaxy Watch Active series, Galaxy Watch series, and Galaxy Fit2 series watches.

Upon information and belief, these products are offered in several varieties, including different options for size, display, processor, storage, and battery. The products listed are exemplary, and Plaintiffs will be able to provide a more comprehensive list after discovery. Upon information and belief, Samsung also infringes one or more claims of the '387 Patent through its use of geofencing or location information in its advertising or content delivery efforts or its own applications it develops.

54. For example, upon information and belief, Samsung's mobile devices infringe at least claim 1 of the '387 Patent. Samsung makes, uses, sells, offers for sale, imports, exports, supplies, or distributes within the United States its mobile devices and thus directly infringes the '387 Patent.

55. Samsung indirectly infringes the '387 Patent as provided by 35 U.S.C. § 271(b) by inducing infringement by others, such as manufacturers, resellers and end-user customers in this District and throughout the United States. For example, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices, who, upon information and belief, perform each step of the claimed invention as directed by Samsung. Samsung received actual notice of the '387 Patent at least as early as the filing of this Complaint.

56. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices, causing Samsung's mobile devices to be manufactured, and providing directions, instructions, schematics, diagrams, or designs to its manufacturers, resellers, or operators/carriers to make or use Samsung's mobile devices in a manner that directly infringes the '387 Patent induce infringement by others, such as manufacturers, resellers, or operators/carriers in this District and throughout the United States. Upon information and belief, through its manufacture and sales of Samsung's mobile devices, Samsung performed the acts that constitute induced infringement with knowledge or willful blindness that the induced acts would constitute infringement.

57. Samsung also indirectly infringes the '387 Patent by contributing to infringement by others, such as manufacturers, resellers, and operators/carriers, in accordance with 35 U.S.C. § 271(c) in this District and throughout the United States. Upon information and belief, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices.


58. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices and causing Samsung's mobile devices to be manufactured and sold contribute to Samsung's manufacturers, resellers, and operators/carriers making or using Samsung's mobile devices in a normal and customary way that infringes the '387 Patent. Upon information and belief, Samsung's mobile devices constitute the material part of Plaintiffs' patented invention, have no substantial non-infringing uses, and are known by Samsung to be especially made or especially adapted for use to infringe the '387 Patent.

59. Samsung provides a mobile device comprising: memory, at least one processor operably coupled to the memory, a location-determination component, and at least one module


configured for execution by the at least one processor, including as demonstrated in the exemplary text and images below:

**SAMSUNG** Mobile TV & Audio Home Appliances Smart Home Computing Offers Collections Explore Support Business


Home / Phones / All Phones




**Galaxy S**  
Starting from \$99.99<sup>8</sup>



**Galaxy Note**  
Starting from \$499.99<sup>8</sup>



**Galaxy Z Fold**  
Starting from \$1449.99<sup>8</sup>



**Galaxy Z Flip**  
Starting from \$649.99<sup>8</sup>

<https://www.samsung.com/us/>

**SAMSUNG** Galaxy S10e | S10 | S10+ | S10 5G

HIGHLIGHTS **SPECS** COMPARE

**Memory**

**Galaxy S10e**

- 6GB RAM with 128GB internal storage
- 8GB RAM with 256GB internal storage

**Galaxy S10**

- 8GB RAM with 128GB internal storage
- 8GB RAM with 512GB internal storage

**Galaxy S10+**

- 8GB RAM with 128GB internal storage
- 8GB RAM with 512GB internal storage (Only ceramic models)
- 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)

**Galaxy S10 5G**

- 8GB RAM with 256GB internal storage
- 8GB RAM with 512GB internal storage

**SAMSUNG** Galaxy S10e | S10 | S10+ | S10 5G

HIGHLIGHTS **SPECS** COMPARE

**AP**

8nm 64-bit Octa-Core Processor ※ 2.73GHz (Maximum Clock Speed) + 2.31GHz + 1.95GHz  
7nm 64-bit Octa-Core Processor ※ 2.84GHz (Maximum Clock Speed) + 2.41GHz + 1.78GHz

\*May differ by country and carrier.

**SAMSUNG** Galaxy S10e | S10 | S10+ | S10 5G

HIGHLIGHTS **SPECS** COMPARE

**OS**

Android 9 (Pie)

SAMSUNG

Galaxy S10e | S10 | S10+ | S10 5G

HIGHLIGHTS | **SPECS** | COMPARE

Network & Connectivity

Enhanced 4x4 MIMO, Up to 7CA, LAA, LTE Cat.20  
Up to 2.0Gbps Download / Up to 150Mbps Upload

5G Non-Standalone (NSA), Sub6 / mmWave (Galaxy S10 5G only)

Wi-Fi 802.11 a/b/g/n/ac/ax (2.4/5GHz), VHT80 MU-MIMO, 1024QAM  
Up to 1.2Gbps Download / Up to 1.2Gbps Upload

Bluetooth® v 5.0 (LE up to 2Mbps), ANT+, USB type-C, NFC, Location (GPS, Galileo, Glonass, BeiDou)

\*Actual speed may vary depending on country, carrier, and user environment.  
\*Depending on country or carrier, 5G connectivity may use Sub6 only, mmWave only, or Sub6 and mmWave together.  
\*Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries.

SAMSUNG

Galaxy S10e | S10 | S10+ | S10 5G

HIGHLIGHTS | **SPECS** | COMPARE

[Bixby](#)

Bixby Routines

Bixby

Bixby Vision

Apps Mode

- Home décor
- Makeup & Hair Dyeing
- Styling

Lens Mode

- Image
- Shopping
- Translation
- Place

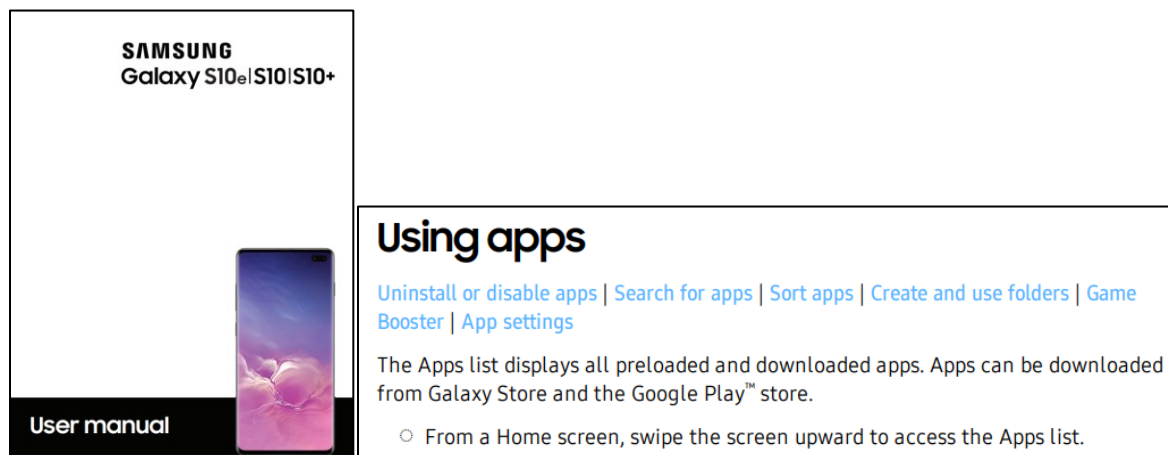
- Picture play
- 3rd party Downloads
- Food
- Wine
- Book
- QR Code

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

60. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: receiving, from an application program during its execution in the mobile device, one or more requests to reserve at least one selected geographic area of interest, wherein the at least one selected geographic area of interest in each of the one or more requests is being requested via said application program to be reserved for having a particular identifier associated with the at least one selected geographic area of interest provided to said application program after



it has been determined, by at least use of the at least one processor and of location information representing at least one physical geographic location of the mobile device as determined by the location-determination component, that the mobile device has at least entered the at least one selected geographic area of interest, and wherein each of the one or more requests comprises data representing a) said particular identifier, represented by a data string, as content provided via said application program to be associated with the at least one selected geographic area of interest, b) at least one latitude value, at least one longitude value, and at least one radius value, each being provided via said application program, to be used for establishing a perimeter boundary for the at least one selected geographic area of interest, and c) information specifying at least one area bound by the perimeter boundary as the at least one selected geographic area of interest, including as demonstrated in the exemplary text and images below:



[https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT\\_SM-G970U\\_G973U\\_G975U\\_EN\\_UM\\_R\\_11.0\\_022221\\_FINAL.pdf](https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT_SM-G970U_G973U_G975U_EN_UM_R_11.0_022221_FINAL.pdf)

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Set up for geofence monitoring

The first step in requesting geofence monitoring is to request the necessary permissions. To use geofencing, your app must request the following:

- `ACCESS_FINE_LOCATION`
- `ACCESS_BACKGROUND_LOCATION` if your app targets Android 10 (API level 29) or higher

To learn more, see the guide on how to [request location permissions](#).

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
  android:allowBackup="true">
  ...
  <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
  android:allowBackup="true">
  ...
  <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

To access the location APIs, you need to create an instance of the Geofencing client. To learn how to connect your client:

Kotlin    Java

```
lateinit var geofencingClient: GeofencingClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this)
}
```

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

## Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
            Java

geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```

geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>

61. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: registering said application program, in the memory, for having said particular identifier provided to said application program after it has been determined, by at least use of the at least one processor and of the location information, that the mobile device has at least entered the at least one selected geographic area of interest, including as demonstrated in the exemplary text and images below:

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.



## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
            Java

geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequest.Builder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   **Java**

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

<https://developer.android.com/training/location/geofencing>

62. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: storing, in the memory, at least one record, for said application program, associated with said particular identifier and the at least one selected geographic area of interest, including as demonstrated in the exemplary text and images below:

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Memory	<b>Galaxy S10e</b>		
	<ul style="list-style-type: none"> <li>• 6GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 256GB internal storage</li> </ul>		
	<b>Galaxy S10+</b>		
	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage (Only ceramic models)</li> <li>• 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)</li> </ul>		
	<b>Galaxy S10</b>		
	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>		
	<b>Galaxy S10 5G</b>		
	<ul style="list-style-type: none"> <li>• 8GB RAM with 256GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>		

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin   Java

```

geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });

```

<https://developer.android.com/training/location/geofencing>

## GeofencingClient

public class **GeofencingClient** extends `GoogleApi<Api.ApiOptions.NoOptions>`

The main entry point for interacting with the geofencing APIs.

Get an instance of this client via `LocationServices.getGeofencingClient(Activity)`.

All methods are thread safe.

### Public Method Summary

Task<Void>	<code>addGeofences(GeofencingRequest geofencingRequest, PendingIntent pendingIntent)</code> Sets alerts to be notified when the device enters or exits one of the specified geofences.
Task<Void>	<code>removeGeofences(List&lt;String&gt; geofenceRequestIds)</code> Removes geofences by their request IDs.
Task<Void>	<code>removeGeofences(PendingIntent pendingIntent)</code> Removes all geofences associated with the given <code>pendingIntent</code> .

```
public Task<Void> addGeofences (GeofencingRequest geofencingRequest, PendingIntent pendingIntent)
```

Sets alerts to be notified when the device enters or exits one of the specified geofences. If an existing geofence with the same request ID is already registered, the old geofence is replaced by the new one, and the new `PendingIntent` is used to generate intents for alerts.

`Task` is completed when geofences are successfully added or failed to be added. Upon failure, an `ApiException` will be set on the `Task`. Refer to `GeofenceStatusCodes` for possible errors when adding geofences.

When a geofence transition (for example, entering or exiting) matches one of the transition filter (see `Geofence.Builder.setTransitionTypes(int)`) in the given geofence list, an intent is generated using the given pending intent. You can call `GeofencingEvent.fromIntent(android.content.Intent)` to get the transition type, geofences that triggered this intent and the location that triggered the geofence transition.

In case network location provider is disabled by the user, the geofence service will stop updating, all registered geofences will be removed and an intent is generated by the provided pending intent. In this case, the `GeofencingEvent` created from this intent represents an error event, where `GeofencingEvent.hasError()` returns `true` and `GeofencingEvent.getErrorCode()` returns `GeofenceStatusCodes.GEOFENCE_NOT_AVAILABLE`.

This method requires `Manifest.permission.ACCESS_FINE_LOCATION`.

#### Parameters

<code>geofencingRequest</code>	geofencing request that include a list of geofences to be added and related triggering behavior. The request must be created using <code>GeofencingRequest.Builder</code> .
<code>pendingIntent</code>	a pending intent that will be used to generate an intent when matched geofence transition is observed

#### Throws

<code>SecurityException</code>	if the app does not have <code>Manifest.permission.ACCESS_FINE_LOCATION</code> permission
<code>NullPointerException</code>	if <code>geofencingRequest</code> or <code>pendingIntent</code> is null

<https://developers.google.com/android/reference/com/google/android/gms/location/GeofencingClient.html>

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin  Java

class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

<https://developer.android.com/training/location/geofencing>

63. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: obtaining the location information representing at least one physical geographic location of the mobile device as determined by the location-determination component, including as demonstrated in the exemplary text and images below:

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G		
		HIGHLIGHTS	SPECS	COMPARE
Network & Connectivity		<p>Enhanced 4x4 MIMO, Up to 7CA, LAA, LTE Cat.20 Up to 2.0Gbps Download / Up to 150Mbps Upload</p> <p>5G Non-Standalone (NSA), Sub6 / mmWave (Galaxy S10 5G only)</p> <p>Wi-Fi 802.11 a/b/g/n/ac/ax (2.4/5GHz), VHT80 MU-MIMO, 1024QAM Up to 1.2Gbps Download / Up to 1.2Gbps Upload</p> <p>Bluetooth® v 5.0 (LE up to 2Mbps), ANT+, USB type-C, NFC, Location (GPS, Galileo, Glonass, BeiDou)</p> <p><small>*Actual speed may vary depending on country, carrier, and user environment. *Depending on country or carrier, 5G connectivity may use Sub6 only, mmWave only, or Sub6 and mmWave together. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries.</small></p>		

<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

## Set up for geofence monitoring

The first step in requesting geofence monitoring is to request the necessary permissions. To use geofencing, your app must request the following:

- `ACCESS_FINE_LOCATION`
- `ACCESS_BACKGROUND_LOCATION` if your app targets Android 10 (API level 29) or higher

To learn more, see the guide on how to [request location permissions](#).

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
  android:allowBackup="true">
  ...
  <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
  android:allowBackup="true">
  ...
  <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

To access the location APIs, you need to create an instance of the Geofencing client. To learn how to connect your client:

Kotlin

Java

```
lateinit var geofencingClient: GeofencingClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this)
}
```

<https://developer.android.com/training/location/geofencing>

64. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: after it has been determined, by at least use of the at least one processor, of the location information, and of the at least one selected geographic area of interest associated with the at least one record stored in the memory for said application program, that the mobile device has at least entered the at least one selected geographic area of interest associated with the at least one record stored in the memory and has remained therein for at least a designated length of time, providing at least said particular identifier associated with the at least one record stored in the memory to said application program, including as demonstrated in the exemplary text and images below:



## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or  *dwell* , before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin    Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin      Java
class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

**Use the dwell transition type to reduce alert spam**

If you receive a large number of alerts when driving briefly past a geofence, the best way to reduce the alerts is to use a transition type of `GEOFENCE_TRANSITION_DWELL` instead of `GEOFENCE_TRANSITION_ENTER`. This way, the dwelling alert is sent only when the user stops inside a geofence for a given period of time. You can choose the duration by setting a `loitering delay`.

<https://developer.android.com/training/location/geofencing>

65. By engaging in the conduct described herein, Samsung has injured Plaintiffs and is thus liable for infringement of the '387 Patent, pursuant to 35 U.S.C. § 271. Samsung has committed these acts of infringement without license or authorization.

66. As a result of Samsung's infringement of the '387 Patent, Plaintiffs have suffered monetary damages and are entitled to a monetary judgment in an amount adequate to compensate for Samsung's past infringement, together with interests and costs. In addition, Samsung's infringement is causing irreparable harm and monetary damage to Plaintiffs and will continue to do so unless and until Samsung is enjoined by the Court.

**COUNT III: CLAIM FOR PATENT INFRINGEMENT OF THE '447 PATENT**

67. Plaintiffs repeat and reallege the allegations in paragraphs 1-66 as if fully set forth herein.

68. Samsung has infringed, contributed to the infringement of, and/or induced infringement of the '447 Patent by making, using, selling, offering for sale, or importing into the United States, or by intending that others make, use, import into, offer for sale, or sell in the United States, products and/or methods covered by one or more claims of the '447 Patent including, but not limited to, mobile devices and components thereof.

69. Upon information and belief, Samsung's mobile devices, including smartphones, tablets, and watches, that infringe one or more claims of the '447 Patent include at least the following products, as well as products with reasonably similar functionality:

- Galaxy S series, Galaxy Note series, Galaxy Z Flip series, Galaxy Fold series, and Galaxy A series smartphones;
- Galaxy Tab S series, Galaxy Tab A series, and Galaxy Book S series tablets; and
- Galaxy Watch 3 series, Galaxy Watch 3 Titanium series, Galaxy Watch Active2 series, Galaxy Watch Active series, Galaxy Watch series, and Galaxy Fit2 series watches.

Upon information and belief, these products are offered in several varieties, including different options for size, display, processor, storage, and battery. The products listed are exemplary, and Plaintiffs will be able to provide a more comprehensive list after discovery. Upon information and belief, Samsung also infringes one or more claims of the '447 Patent through its use of geofencing or location information in its advertising or content delivery efforts or its own applications it develops.

70. For example, upon information and belief, Samsung's mobile devices infringe at least claim 1 of the '447 Patent. Samsung makes, uses, sells, offers for sale, imports, exports, supplies, or distributes within the United States its mobile devices and thus directly infringes the '447 Patent.

71. Samsung indirectly infringes the '447 Patent as provided by 35 U.S.C. § 271(b) by inducing infringement by others, such as manufacturers, resellers, and end-user customers in this

District and throughout the United States. For example, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices, who, upon information and belief, perform each step of the claimed invention as directed by Samsung. Samsung received actual notice of the '447 Patent at least as early as the filing of this Complaint.

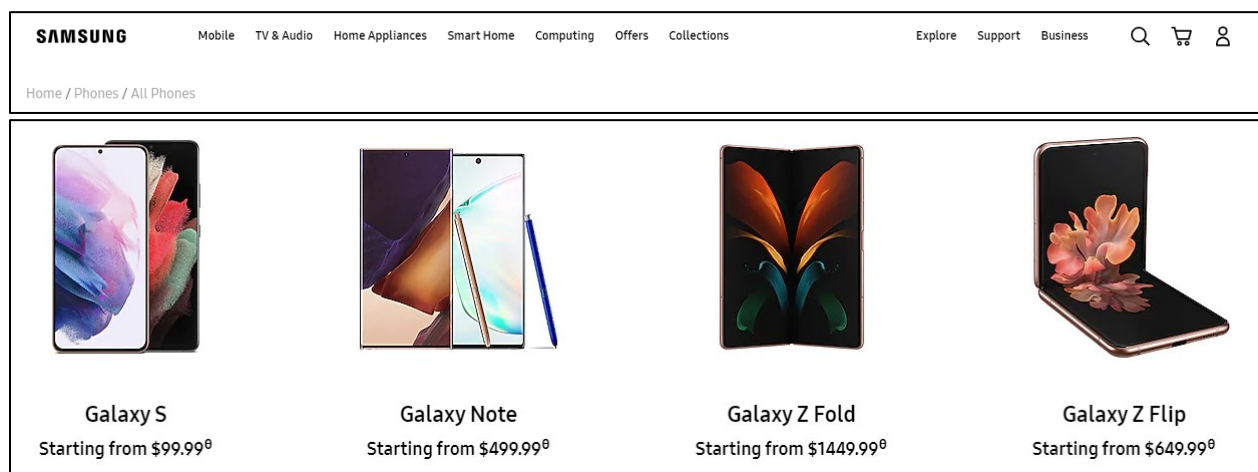
72. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices, causing Samsung's mobile devices to be manufactured, and providing directions, instructions, schematics, diagrams, or designs to its manufacturers, resellers, or operators/carriers to make or use Samsung's mobile devices in a manner that directly infringes the '447 Patent induce infringement by others, such as manufacturers, resellers, or operators/carriers in this District and throughout the United States. Upon information and belief, through its manufacture and sales of Samsung's mobile devices, Samsung performed the acts that constitute induced infringement with knowledge or willful blindness that the induced acts would constitute infringement.

73. Samsung also indirectly infringes the '447 Patent by contributing to infringement by others, such as manufacturers, resellers, and operators/carriers, in accordance with 35 U.S.C. § 271(c) in this District and throughout the United States. Upon information and belief, direct infringement is the result of activities performed by manufacturers, resellers, or operators/carriers of Samsung's mobile devices.

74. Upon information and belief, Samsung's affirmative acts of selling Samsung's mobile devices and causing Samsung's mobile devices to be manufactured and sold contribute to Samsung's manufacturers, resellers, and operators/carriers making or using Samsung's mobile devices in a normal and customary way that infringes the '447 Patent. Upon information and belief,

Samsung's mobile devices constitute the material part of Plaintiffs' patented invention, have no substantial non-infringing uses, and are known by Samsung to be especially made or especially adapted for use to infringe the '447 Patent.

75. Samsung provides a mobile device comprising: memory, at least one processor operably coupled to the memory, a location-determination component, and at least one module configured for execution by the at least one processor, including as demonstrated in the exemplary text and images below:




<https://www.samsung.com/us/>

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G	
		HIGHLIGHTS	SPECS
Memory	<b>Galaxy S10e</b>	<ul style="list-style-type: none"> <li>• 6GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 256GB internal storage</li> </ul>	
	<b>Galaxy S10</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	
	<b>Galaxy S10+</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 128GB internal storage</li> <li>• 8GB RAM with 512GB internal storage (Only ceramic models)</li> <li>• 12GB RAM with 1TB internal storage (Only Performance Edition ceramic models)</li> </ul>	
	<b>Galaxy S10 5G</b>	<ul style="list-style-type: none"> <li>• 8GB RAM with 256GB internal storage</li> <li>• 8GB RAM with 512GB internal storage</li> </ul>	

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G
		HIGHLIGHTS <u>SPECS</u> COMPARE
AP	8nm 64-bit Octa-Core Processor ※ 2.730GHz (Maximum Clock Speed) + 2.310GHz + 1.950GHz 7nm 64-bit Octa-Core Processor ※ 2.840GHz (Maximum Clock Speed) + 2.410GHz + 1.780GHz  *May differ by country and carrier.	

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G
		HIGHLIGHTS <u>SPECS</u> COMPARE
OS	Android 9 (Pie)	

SAMSUNG		Galaxy S10e   S10   S10+   S10 5G
		HIGHLIGHTS <u>SPECS</u> COMPARE
Network & Connectivity	Enhanced 4x4 MIMO, Up to 7CA, LAA, LTE Cat.20 Up to 2.0Gbps Download / Up to 150Mbps Upload  5G Non-Standalone (NSA), Sub6 / mmWave (Galaxy S10 5G only)  Wi-Fi 802.11 a/b/g/n/ac/ax (2.4/5GHz), VHT80 MU-MIMO, 1024QAM Up to 1.2Gbps Download / Up to 1.2Gbps Upload  Bluetooth® v 5.0 (LE up to 2Mbps), ANT+, USB type-C, NFC, Location (GPS, Galileo, Glonass, BeiDou)  *Actual speed may vary depending on country, carrier, and user environment. *Depending on country or carrier, 5G connectivity may use Sub6 only, mmWave only, or Sub6 and mmWave together. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries.	

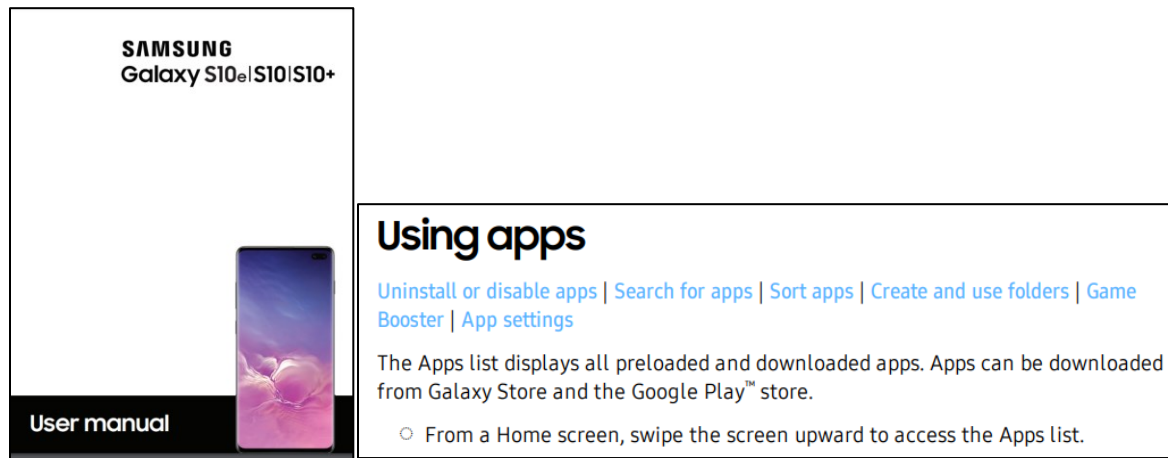
SAMSUNG		Galaxy S10e   S10   S10+   S10 5G
		HIGHLIGHTS <u>SPECS</u> COMPARE
Bixby 	Bixby Routines  Bixby  Bixby Vision  Apps Mode <ul style="list-style-type: none"> <li>• Home décor</li> <li>• Makeup &amp; Hair Dyeing</li> <li>• Styling</li> </ul> Lens Mode <ul style="list-style-type: none"> <li>• Image</li> <li>• Shopping</li> <li>• Translation</li> <li>• Place</li> </ul>	
		<ul style="list-style-type: none"> <li>• Picture play</li> <li>• 3rd party Downloads</li> </ul>
		<ul style="list-style-type: none"> <li>• Food</li> <li>• Wine</li> <li>• Book</li> <li>• QR Code</li> </ul>



<https://www.samsung.com/global/galaxy/galaxy-s10/specs/>

76. Samsung provides a mobile device, wherein the at least one module comprises at least one instruction for: in response to receiving, from an application program during its execution in the mobile device, a request for having at least a first identifier being associated with a first designated geographic area of interest provided to the application program after it has been determined, by at least use of the at least one processor and of location information representing at least one physical geographic location of the mobile device as determined by the location-determination component, that the mobile device has at least entered the first designated geographic area of interest and has remained therein for at least a predetermined length of time, after receiving, from the application program, both a first data string comprising the first identifier to be associated with the first designated geographic area of interest and first geographic data to be used for defining the first designated geographic area of interest and to be used with the first identifier to establish, in the memory, the first designated geographic area of interest as a first area specifically reserved for delivery of the first identifier related to the application program, after establishing, in the memory, the first designated geographic area of interest as the first area specifically reserved for delivery of the first identifier related to the application program, after storing, in the memory, a first record comprising the first identifier, and after determining, by at least use of the at least one processor and of the location information, that the mobile device has at least entered the first area specifically reserved for delivery of the first identifier related to the application program and has remained therein for at least the predetermined length of time, delivering, from the first record stored in the memory and to the application program, at least first

datum comprising at least the first identifier, including as demonstrated in the exemplary text and images below:



[https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT\\_SM-G970U\\_G973U\\_G975U\\_EN\\_UM\\_R\\_11.0\\_022221\\_FINAL.pdf](https://downloadcenter.samsung.com/content/UM/202103/20210304004310357/ATT_SM-G970U_G973U_G975U_EN_UM_R_11.0_022221_FINAL.pdf)

## Create and monitor geofences

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

## Set up for geofence monitoring

The first step in requesting geofence monitoring is to request the necessary permissions. To use geofencing, your app must request the following:

- `ACCESS_FINE_LOCATION`
- `ACCESS_BACKGROUND_LOCATION` if your app targets Android 10 (API level 29) or higher

To learn more, see the guide on how to [request location permissions](#).

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
    android:allowBackup="true">
    ...
    <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
    android:allowBackup="true">
    ...
    <receiver android:name=".GeofenceBroadcastReceiver"/>
</application>
```

To access the location APIs, you need to create an instance of the Geofencing client. To learn how to connect your client:

Kotlin    Java

```
lateinit var geofencingClient: GeofencingClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this)
}
```

## Create and add geofences

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

★ **Note:** On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

### Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
Kotlin      Java
            Java

geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

## Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequestBuilder` class to specify the geofences to monitor and to set how related geofence events are triggered:

Kotlin

Java

```
private GeofencingRequest getGeofencingRequest() {  
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
    builder.addGeofences(geofenceList);  
    return builder.build();  
}
```

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

## Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

Kotlin   Java

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
            FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
}
```

## Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

Kotlin Java

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```

## Handle geofence transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.



The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```

Kotlin  Java

class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER |
            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val geofenceTransitionDetails = getGeofenceTransitionDetails(
                this,
                geofenceTransition,
                triggeringGeofences
            )

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails)
            Log.i(TAG, geofenceTransitionDetails)
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                geofenceTransition))
        }
    }
}

```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

### Use the dwell transition type to reduce alert spam

If you receive a large number of alerts when driving briefly past a geofence, the best way to reduce the alerts is to use a transition type of `GEOFENCE_TRANSITION_DWELL` instead of `GEOFENCE_TRANSITION_ENTER`. This way, the dwelling alert is sent only when the user stops inside a geofence for a given period of time. You can choose the duration by setting a `loitering delay`.

<https://developer.android.com/training/location/geofencing>

77. By engaging in the conduct described herein, Samsung has injured Plaintiffs and is thus liable for infringement of the '447 Patent, pursuant to 35 U.S.C. § 271. Samsung has committed these acts of infringement without license or authorization.

78. As a result of Samsung's infringement of the '447 Patent, Plaintiffs have suffered monetary damages and are entitled to a monetary judgment in an amount adequate to compensate for Samsung's past infringement, together with interests and costs. In addition, Samsung's infringement is causing irreparable harm and monetary damage to Plaintiffs and will continue to do so unless and until Samsung is enjoined by the Court.

**DEMAND FOR JURY TRIAL**

79. Plaintiffs hereby demand a trial by jury on all claims so triable.

**PRAYER FOR RELIEF**

WHEREFORE, Plaintiffs respectfully request that this Court enter judgment in their favor and grant the following relief:

- A. Adjudge that Samsung infringes the Asserted Patents;
- B. Adjudge that the Asserted Patents are valid and enforceable;
- C. Award Plaintiffs damages in an amount adequate to compensate Plaintiffs for Samsung's infringement of the Asserted Patents, but in no event less than a reasonable royalty under 35 U.S.C. § 284;
- D. Award enhanced damages pursuant to 35 U.S.C. § 284;
- E. Award Plaintiffs pre-judgment and post-judgment interest to the full extent allowed under the law, as well as their costs;

- F. Enter an order finding that this is an exceptional case and awarding Plaintiffs their reasonable attorneys' fees pursuant to 35 U.S.C. § 285;
- G. Enter a permanent injunction against all Samsung products found to infringe the Asserted Patents;
- H. Award, in lieu of an injunction, a compulsory forward royalty;
- I. Order an accounting of damages; and
- J. Award such other relief, including equitable relief, as the Court may deem appropriate and just under the circumstances.

**DATED:** July 30, 2021

Respectfully submitted,

/s/ Thomas M. Melsheimer

**Thomas M. Melsheimer**

Texas Bar No. 13922550

tmelsheimer@winston.com

**M. Brett Johnson**

Texas Bar No. 00790975

mbjohnson@winston.com

**Rex A. Mann**

Texas Bar No. 24075509

rmann@winston.com

**Chad B. Walker**

Texas Bar No. 24056484

cbwalker@winston.com

**Ahtoosa A. Dale**

Texas Bar No. 24101443

adale@winston.com

**WINSTON & STRAWN LLP**

2121 North Pearl Street, Suite 900

Dallas, TX 75201

Telephone: (214) 453-6500

**Saranya Raghavan** (*pro hac vice*)

sraghavan@winston.com

**WINSTON & STRAWN LLP**

35 West Wacker Drive

Chicago, IL 60601

Telephone: (312) 558-5600

**Matt Hopkins** (*pro hac vice*)

mhopkins@winston.com

**WINSTON & STRAWN LLP**

1901 L Street NW

Washington, D.C. 20036

Telephone: (202) 282-5000

**ATTORNEYS FOR PLAINTIFFS**